Package ‘cghRA’

March 3, 2017

Type Package

Title Array CGH Data Analysis and Visualization

Version 1.6.0

Date 2017-03-03

Author Sylvain Mareschal

Maintainer Sylvain Mareschal <maressyl@gmail.com>

URL http://www.ovsa.fr/cghRA

BugReports https://github.com/maressyl/R.cghRA/issues

Description Provides functions to import data from Agilent CGH arrays and process them according to the cghRA workflow. Implements several algorithms such as WACA, STEPS and cnvScore and an interactive graphical interface.

License GPL (>= 3)

Depends methods, Rgb (>= 1.5.0), R (>= 2.10)

Imports DNAcopy, utils, stats

Suggests tcltk, tkrplot, parallel, GLAD, graphics, grDevices

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-03-03 21:25:45

R topics documented:

bias 2
cghRA.array 4
cghRA.array-class 4
cghRA.copies-class 7
cghRA.copies-constructor 10
cghRA.design-class 11
cghRA.design-constructor 14
cghRA.probes-class 15
cghRA.probes-constructor 17

cghRA.regions-class 17
cghRA.regions-constructor 21
cghRA.series 22
cghRA.series-class 22
cnvScore 24
copies 26
Design file parser 28
drawableFromClass.cghRA.probes 29
fillGaps 30
GEDI 31
localize 32
map2design 34
model.apply 35
model.auto 38
model.test 41
parallelize 44
parseKaryo 45
penetrance 46
Probe file parser 47
process 48
segmentMap-class 52
SRA & LRA 53
STEPS 55
tk.annotate 56
tk.cghRA 57
tk.design 58
tk.modelize 59
tk.series 60
tk.value 61
trace2track 62
track.CNV.DGVsupp 62
WACA 63

Index **65**

`bias` *WACA bias computation for a probe series*

Description

This function computes the various probe-dependant biases used by the Waves aCGH Correction Algorithm (WACA), in order to apply this correction to CGH arrays using these probes.

Usage

```
bias(chromFiles, probeChrom, probeStarts, probeEnds,
  chromPattern = "^(.+)\\.[^\\.]+$", fragSites = c(AluI = "AG|CT", RsaI = "GT|AC"),
   digits = 6, verbose = 1)
```

Arguments

`chromFiles`	Character vector, paths to chromosome sequences (a single fasta file for each chromosome).
`probeChrom`	`character` vector, for each probe its chromosome location.
`probeStarts`	`integer` vector, for each probe its chromosome starting position (first base is 1, starting position is comprised in the probe).
`probeEnds`	`integer` vector, for each probe its chromosome ending position (first base is 1, ending position is comprised in the probe).
`chromPattern`	Single character value, a regular expression to be used for chromosome name extraction from `chromFiles`. It needs to capture a single value for replacement, default value will use the base names of the files without extension as chromosome names.
`fragSites`	Named character vector describing the restriction enzymes used for the CGH experiment. Restriction sites are described in upper cases, with a pipe at the fragmentation position (see the default value for an example). Only A, C, G and T letters allowed.
`digits`	Single integer value, to be passed to `round` for all bias values.
`verbose`	Single integer value, whether to print diagnostic `messages` or not.

Value

Returns a double matrix, with probes in rows and the following columns :

`wGC150`	GC frequency in a window of 150 kb on each side of the probe
`wGC500`	GC frequency in a window of 500 kb on each side of the probe
`wGCprobe`	GC frequency in the probe sequence
`wGCfrag`	GC frequency in the fragment holding the probe
`wFragSize`	Size (in bp) of the fragment holding the probe

Author(s)

Sylvain Mareschal

References

Lepretre F. et al. (2010) Waved aCGH: to smooth or not to smooth. Nucleic Acids Res. 2010 Apr;38(7):e94

See Also

`WACA`, `localize`

cghRA.array *cghRA.array class constructor*

Description

This function returns a new cghRA.array object from various arguments.

Usage

```
cghRA.array(.design, .probes, .name, .parameters, warn = TRUE)
```

Arguments

.design	An object of class cghRA.design, as returned by the cghRA.design constructor.
.probes	An object of class cghRA.probes, as returned by the cghRA.probes constructor.
.name	Single character value, to fill the name field inherited from drawable.
.parameters	A list of drawing parameters, to fill the parameters field of the object.
warn	Single logical value, to be passed to the cghRA.array-class check method.

Value

An object of class cghRA.array.

Author(s)

Sylvain Mareschal

See Also

cghRA.array-class

cghRA.array-class *Class* "cghRA.array"

Description

This class is the main component of the cghRA object-oriented package. Each CGH array must be stored in a distinct cghRA.array object.

Objects from this class should always be produced by the cghRA.array constructor.

This class is a hub, it provides methods to apply various CGH analysis tools in a straight-forward way.

The Reference Class system is used notably to share designs objects between arrays, as several arrays may have values for the same probes.

Extends

Class `crossable`, directly.
Class `sliceable`, by class `crossable`, distance 2.
Class `drawable`, by class `crossable`, distance 3.

All reference classes extend and inherit methods from `envRefClass`.

Fields

`assembly`**:** Single `character` value, the assembly version for the coordinates stored in the object. Must have length 1, should not be `NA`.

`design`**:** Object of class `cghRA.design`

`organism`**:** Single `character` value, the name of the organism whose data is stored in the object. Must have length 1, should not be `NA`.

`probes`**:** Object of class `cghRA.probes`.

The following fields are inherited (from the corresponding class):

- name (drawable)
- parameters (drawable)

Methods

`as.CNA()`**:** Returns a CNA object (DNAcopy) with the object content.

`as.profileCGH(chrom = , quiet = )`**:** Returns a profileCGH object (GLAD) with the object content.
- **chrom** : single character value defining how to deal with chromosome names :
'merged' forces chromosome arms to be merged (as chromosome arms are not handled)
'levels' converts chromosome to integers (can be deceiving for factors)
- **quiet** : single logical value, whether to warn for factor to integer conversion or not.

`DLRS(method = , na.rm = )`**:** Computes the Derivative Log Ratio Spread from the probes.
- **method** : 'agilent' or 'original', implying distinct formulas.

`DNAcopy(smooth = , ...)`**:** Apply the Circular Binary Segmentation, as implemented in DNA-copy, and return a cghRA.regions object.
- **smooth** : a list of arguments to be passed to smooth.CNA(), TRUE to use the default parameters or FALSE to skip smoothing.
- **...** : arguments to be passed to segment().

`extract(i = , j = )`**:** Extracts values from 'probes' and 'design' into a data.frame.
- **i** : row selection, see the R5Table method for further details.
- **j** : column selection, see the R5Table method for further details.

`GADA(...)`**:** Apply the Genome Alteration Detection Analysis, as implemented in GADA, and return a cghRA.regions object.
- **smooth** : a list of arguments to be passed to smooth.CNA(), TRUE to use the default parameters or FALSE to skip smoothing.
- **...** : arguments to be passed to segment().

`GLAD(chrom = , quiet = , output = , ...)`: Apply the Gain and Loss Analysis of Dna, as implemented in GLAD, and return a cghRA.regions object.
- **chrom, quiet** : to be passed to the as.profileCGH method.
- **output** : single character value defining the returned value :
'regions' returns a cghRA.regions object with the segmented genome
'raw' returns the glad() output
'both' adds a 'cghRA.regions' element to the glad() output list to return both
- **...** : arguments to be passed to glad().

`MAplot(pch = , cex = , xlab = , ylab = , ...)`: MA plot of all the probes.
- **...** : arguments to be passed to plot().

`maskByFlag(flags = , pattern = , multiple = , na = )`: Replaces logRatios of flagged probes by NA.
- **flags** : character vector, the columns to coerce as boolean and use as flags.
- **pattern** : single logical value, whether to consider 'flags' as regular expressions or fixed values.
- **multiple** : mask a probe when 'all' its flag columns are TRUE or when 'any' is.

`replicates(fun = , na.rm = , ...)`: Apply 'fun' to replicated probes (same name), masking all members but one.
- **fun** : single character value, the function to apply.
- **...** : to be passed to 'fun'.

`spatial(filename = , palSize = , palEnds = , ...)`: Produces a spatial representation of the logRatios, to identify spatial biases.
- **filename** : single character value, the path to the PNG output.
- **palSize** : single integer value, the amount of color levels for logRatios. Should be lesser or equal to 254 to produce small PNG files.
- **palEnds** : character vector to be passed to colorRampPalette() for palette generation.

`WACA()`: Apply the Waves aCGH Correction Algorithm (Lepretre et al. 2009) to the array logRatios.

The following methods are inherited (from the corresponding class):

- callParams (drawable)
- callSuper (envRefClass)
- check (drawable, overloaded)
- chromosomes (drawable, overloaded)
- copy (envRefClass)
- cross (crossable)
- defaultParams (sliceable, overloaded)
- draw (sliceable)
- export (envRefClass)
- field (envRefClass)
- fix.param (drawable)
- getChromEnd (sliceable, overloaded)
- getClass (envRefClass)

- getName (drawable)
- getParam (drawable)
- getRefClass (envRefClass)
- import (envRefClass)
- initFields (envRefClass)
- initialize (drawable, overloaded)
- setName (drawable)
- setParam (drawable)
- show (sliceable, overloaded)
- slice (sliceable, overloaded)
- trace (envRefClass)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.array`

`cghRA.series-class`, `cghRA.design-class`, `cghRA.probes-class`, `cghRA.regions-class`

`cghRA.copies-class` *Class* `"cghRA.copies"`

Description

This class is derived from `cghRA.regions`, whose `model.apply` method is the commonest way to obtain `cghRA.copies` objects.

Extends

Class `cghRA.regions`, directly.
Class `track.table`, by class `cghRA.regions`, distance 2.
Class `refTable`, by class `cghRA.regions`, distance 3.
Class `crossable`, by class `cghRA.regions`, distance 3.
Class `sliceable`, by class `cghRA.regions`, distance 4.
Class `drawable`, by class `cghRA.regions`, distance 5.

All reference classes extend and inherit methods from `envRefClass`.

Fields

The following fields are inherited (from the corresponding class):

- assembly (track.table)
- checktrack (track.table)
- colCount (refTable)
- colIterator (refTable)
- colNames (refTable)
- colReferences (refTable)
- index (track.table)
- model (cghRA.regions)
- modelizeCall (cghRA.regions)
- name (drawable)
- organism (track.table)
- parameters (drawable)
- rowCount (refTable)
- rowNamed (refTable)
- rowNames (refTable)
- segmentCall (cghRA.regions)
- sizetrack (track.table)
- subtrack (track.table)
- values (refTable)

Methods

The following methods are inherited (from the corresponding class):

- addArms (track.table)
- addColumn (track.table)
- addDataFrame (refTable)
- addEmptyRows (refTable)
- addList (track.table)
- addVectors (refTable)
- buildCalls (track.table)
- buildGroupPosition (track.table)
- buildGroupSize (track.table)
- buildIndex (track.table)
- callParams (drawable)
- callSuper (envRefClass)
- check (cghRA.regions, overloaded)

- chromosomes (track.table)
- coerce (track.table)
- colOrder (refTable)
- copy (refTable)
- cross (crossable)
- defaultParams (cghRA.regions, overloaded)
- delColumns (track.table)
- draw (sliceable)
- erase (refTable)
- eraseArms (track.table)
- export (envRefClass)
- extract (refTable)
- field (envRefClass)
- fill (track.table)
- fillGaps (cghRA.regions)
- fix.param (drawable)
- getChromEnd (track.table)
- getClass (envRefClass)
- getColCount (refTable)
- getColNames (refTable)
- getLevels (refTable)
- getName (drawable)
- getParam (drawable)
- getRefClass (envRefClass)
- getRowCount (refTable)
- getRowNames (refTable)
- import (envRefClass)
- indexes (refTable)
- initFields (envRefClass)
- initialize (cghRA.regions)
- isArmed (track.table)
- karyotype (cghRA.regions)
- metaFields (track.table)
- model.apply (cghRA.regions)
- model.auto (cghRA.regions)
- modelized (cghRA.regions)
- model.test (cghRA.regions)

- proportions (cghRA.regions)
- rowOrder (track.table)
- segMerge (track.table)
- segOverlap (track.table)
- setColNames (track.table)
- setLevels (track.table)
- setName (drawable)
- setParam (drawable)
- setRowNames (refTable)
- show (cghRA.regions, overloaded)
- size (track.table)
- slice (track.table)
- status (cghRA.regions)
- trace (envRefClass)
- types (refTable)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.regions-class`

`cghRA.copies-constructor` *cghRA.copies class constructor*

Description

This function returns a new `cghRA.copies` object from various arguments.

Notice the `new()` alternative can be used to produce an empty object, setting only the fields not the content.

Usage

```
cghRA.copies(..., warn = TRUE)
```

Arguments

`...`	Arguments to be passed through the inherited constructors up to `refTable`.
`warn`	Single logical value, to be passed to the `cghRA.copies` check method.

Value

An object of class `cghRA.copies`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.copies-class`, `cghRA.regions-class`, `track.table-class`, `refTable-class`

`cghRA.design-class` *Class* `"cghRA.design"`

Description

This class is part of the `cghRA.array` class. A single object from this class is used to store informations about probes for series of arrays sharing the same CGH design, in order to store only array-specific values in the array variables.

Objects from this class can be produced by the `cghRA.design`, `Agilent.design` and `custom.design` constructors. Alternatively they can be produced by the interactive function `tk.design`, included in `tk.cghRA`.

Extends

Class `track.table`, directly.
Class `refTable`, by class `track.table`, distance 2.
Class `crossable`, by class `track.table`, distance 2.
Class `sliceable`, by class `track.table`, distance 3.
Class `drawable`, by class `track.table`, distance 4.

All reference classes extend and inherit methods from `envRefClass`.

Fields

The following fields are inherited (from the corresponding class):

- assembly (track.table)
- checktrack (track.table)
- colCount (refTable)
- colIterator (refTable)

- colNames (refTable)
- colReferences (refTable)
- index (track.table)
- name (drawable)
- organism (track.table)
- parameters (drawable)
- rowCount (refTable)
- rowNamed (refTable)
- rowNames (refTable)
- sizetrack (track.table)
- subtrack (track.table)
- values (refTable)

Methods

`bias(...)`**:** Computes the Waves aCGH Correction Algorithm (Lepretre et al. 2009) bias for the current design.
- **...** : arguments to be passed to the bias() function (except from 'probeChrom', 'probeStarts' and 'probeEnds').

`remap(...)`**:** Recomputes the coordinates of the probes from the probes and genome sequences. Forces 'chrom' to factor, keeping levels if available.
- **...** : arguments to be passed to the localize() function.

The following methods are inherited (from the corresponding class):

- addArms (track.table)
- addColumn (track.table)
- addDataFrame (refTable)
- addEmptyRows (refTable)
- addList (track.table)
- addVectors (refTable)
- buildCalls (track.table)
- buildGroupPosition (track.table)
- buildGroupSize (track.table)
- buildIndex (track.table)
- callParams (drawable)
- callSuper (envRefClass)
- check (track.table, overloaded)
- chromosomes (track.table)
- coerce (track.table)
- colOrder (refTable)

- copy (refTable)
- cross (crossable)
- defaultParams (track.table, overloaded)
- delColumns (track.table)
- draw (sliceable)
- erase (refTable)
- eraseArms (track.table)
- export (envRefClass)
- extract (refTable)
- field (envRefClass)
- fill (track.table)
- fix.param (drawable)
- getChromEnd (track.table)
- getClass (envRefClass)
- getColCount (refTable)
- getColNames (refTable)
- getLevels (refTable)
- getName (drawable)
- getParam (drawable)
- getRefClass (envRefClass)
- getRowCount (refTable)
- getRowNames (refTable)
- import (envRefClass)
- indexes (refTable)
- initFields (envRefClass)
- initialize (track.table, overloaded)
- isArmed (track.table)
- metaFields (track.table)
- rowOrder (track.table)
- segMerge (track.table)
- segOverlap (track.table)
- setColNames (track.table)
- setLevels (track.table)
- setName (drawable)
- setParam (drawable)
- setRowNames (refTable)
- show (track.table, overloaded)

- size (track.table)
- slice (track.table)
- trace (envRefClass)
- types (refTable)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.design`, `Agilent.design`, `custom.design`, `tk.design`

`cghRA.array-class`, `refTable-class`

`cghRA.design-constructor` *cghRA.design class constructor*

Description

This function returns a new `cghRA.design` object from various arguments.

Notice the `new()` alternative can be used to produce an empty object, setting only the fields not the content.

Usage

```
cghRA.design(..., warn = TRUE)
```

Arguments

`...`	Arguments to be passed through the inherited constructors up to `refTable`.
`warn`	Single logical value, to be passed to the `cghRA.design` check method.

Value

An object of class `cghRA.design`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.design-class`, `track.table-class`, `refTable-class`

`Agilent.design`

cghRA.probes-class *Class* "cghRA.probes"

Description

This class is part of the cghRA.array class, designed to store all probe-related values of a single CGH array.

Objects from this class can be produced by the cghRA.array constructor or by the process function, its interfaces tk.process and tk.cghRA or their sub-functions.

Extends

Class refTable, directly.

All reference classes extend and inherit methods from envRefClass.

Fields

name: Custom name for the object, as a character vector of length 1.

The following fields are inherited (from the corresponding class):

- colCount (refTable)
- colIterator (refTable)
- colNames (refTable)
- colReferences (refTable)
- rowCount (refTable)
- rowNamed (refTable)
- rowNames (refTable)
- values (refTable)

Methods

The following methods are inherited (from the corresponding class):

- addColumn (refTable)
- addDataFrame (refTable)
- addEmptyRows (refTable)
- addList (refTable)
- addVectors (refTable)
- callSuper (envRefClass)
- check (refTable, overloaded)
- coerce (refTable)
- colOrder (refTable)

- copy (refTable)
- delColumns (refTable)
- erase (refTable)
- export (envRefClass)
- extract (refTable)
- field (envRefClass)
- fill (refTable)
- getClass (envRefClass)
- getColCount (refTable)
- getColNames (refTable)
- getLevels (refTable)
- getRefClass (envRefClass)
- getRowCount (refTable)
- getRowNames (refTable)
- import (envRefClass)
- indexes (refTable)
- initFields (envRefClass)
- initialize (refTable, overloaded)
- metaFields (refTable)
- rowOrder (refTable)
- setColNames (refTable)
- setLevels (refTable)
- setRowNames (refTable)
- show (refTable, overloaded)
- trace (envRefClass)
- types (refTable)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.array-class`, `refTable-class`, `tk.process`

`cghRA.probes-constructor` *cghRA.probes class constructor*

Description

This function returns a new `cghRA.probes` object from various arguments.

Notice the `new()` alternative can be used to produce an empty object, setting only the fields not the content.

Usage

```
cghRA.probes(..., .name, warn = TRUE)
```

Arguments

`...`	Arguments to be passed through the inherited constructors up to `refTable`.
`.name`	Single character value, a custom name for the object (for printing purpose essentially).
`warn`	Single logical value, to be passed to the `cghRA.probes` check method.

Value

An object of class `cghRA.probes`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.probes-class`, `refTable-class`

`Agilent.probes`

`cghRA.regions-class` *Class* `"cghRA.regions"`

Description

This class is intended to store a list of genomic segments produced by a segmentation algorithm, with a mean log-ratio for each segment.

Objects from this class are intended to be produced by the `DNAcopy` method of the `cghRA.array` class, or the `cghRA.regions` constructor. Producing such objects is part of the `process` function and its interfaced version `tk.process`, found in `tk.cghRA`.

Extends

Class `track.table`, directly.
Class `refTable`, by class `track.table`, distance 2.
Class `crossable`, by class `track.table`, distance 2.
Class `sliceable`, by class `track.table`, distance 3.
Class `drawable`, by class `track.table`, distance 4.

All reference classes extend and inherit methods from `envRefClass`.

Fields

`model`**:** Numeric vector, storing the parameters and fitness of a copy-number model. See `model.auto` for further details on the components.

`modelizeCall`**:** The R `call` which produced the stored copy-number model.

`segmentCall`**:** The R `call` which produced the segments stored in the object.

The following fields are inherited (from the corresponding class):

- assembly (track.table)
- checktrack (track.table)
- colCount (refTable)
- colIterator (refTable)
- colNames (refTable)
- colReferences (refTable)
- index (track.table)
- name (drawable)
- organism (track.table)
- parameters (drawable)
- rowCount (refTable)
- rowNamed (refTable)
- rowNames (refTable)
- sizetrack (track.table)
- subtrack (track.table)
- values (refTable)

Methods

`fillGaps(...)`**:** Apply the fillGaps() function to extend regions in order to fill inter-segment gaps.

`karyotype(bandTrack, value = , thresholds = , precision = )`**:** Returns a karyotype formula of altered regions.
- **bandTrack** : a track.table object, as returned by track.UCSC_bands().
- **value** : column to use to select altered regions.
- **thresholds** : length 2 numeric vector defining altered values.
- **precision** : single integer value from 1 to 4, amount of digits to consider in banding.

`model.apply(...)`**:** Call the model.apply() function to produce a cghRA.copies object with predicted copy number for each region.

`model.auto(save = , ...)`**:** Call the model.auto() function to automatically fit a copy-number prediction model.
- **save** : single logical value, whether to save the model or only return it

`modelized()`**:** Does the object embed a complete model or not

`model.test(...)`**:** Call the model.test() function to plot the current copy-number model.

`proportions(chrom = , value = , states = , mode = )`**:** Returns the proportion of the chromosomes in given states (in bp involved).
- **chrom** : character vector, chromosome location of the regions to query. Consider track.table$eraseArms() to focus on chromosome arms.
- **value** : single character value, name of the column to use for state assignation.
- **states** : list of states, see penetrance help page for details.

`status(chrom, start, end, value = , na = , fuzzy = , states = )`**:** Returns the copy states in various windows, mimicing penetrance behavior.
- **chrom** : character vector, chromosome location of the regions to query.
- **start** : integer vector, starting position on the chromosome for the regions to query.
- **end** : integer vector, ending position on the chromosome for the regions to query.
- **value** : single character value, name of the column to use for state assignation.
- **na** : single character value, see penetrance() help page for details ('false' is not handled).
- **fuzzy** : single logical value, whether to assign the state when some sub-regions are out or not.
- **states** : list of states, see penetrance help page for details.

The following methods are inherited (from the corresponding class):

- addArms (track.table)
- addColumn (track.table)
- addDataFrame (refTable)
- addEmptyRows (refTable)
- addList (track.table)
- addVectors (refTable)
- buildCalls (track.table)
- buildGroupPosition (track.table)
- buildGroupSize (track.table)
- buildIndex (track.table)
- callParams (drawable)
- callSuper (envRefClass)
- check (track.table, overloaded)
- chromosomes (track.table)
- coerce (track.table)
- colOrder (refTable)
- copy (refTable)

- cross (crossable)
- defaultParams (track.table, overloaded)
- delColumns (track.table)
- draw (sliceable)
- erase (refTable)
- eraseArms (track.table)
- export (envRefClass)
- extract (refTable)
- field (envRefClass)
- fill (track.table)
- fix.param (drawable)
- getChromEnd (track.table)
- getClass (envRefClass)
- getColCount (refTable)
- getColNames (refTable)
- getLevels (refTable)
- getName (drawable)
- getParam (drawable)
- getRefClass (envRefClass)
- getRowCount (refTable)
- getRowNames (refTable)
- import (envRefClass)
- indexes (refTable)
- initFields (envRefClass)
- initialize (track.table, overloaded)
- isArmed (track.table)
- metaFields (track.table)
- rowOrder (track.table)
- segMerge (track.table)
- segOverlap (track.table)
- setColNames (track.table)
- setLevels (track.table)
- setName (drawable)
- setParam (drawable)
- setRowNames (refTable)
- show (track.table, overloaded)
- size (track.table)

- slice (track.table)
- trace (envRefClass)
- types (refTable)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.array-class`, `process`, `tk.process`, `refTable-class`

`cghRA.regions-constructor` *cghRA.regions class constructor*

Description

This function returns a new `cghRA.regions` object from various arguments.

Notice the `new()` alternative can be used to produce an empty object, setting only the fields not the content.

Usage

```
cghRA.regions(..., .model, warn = TRUE)
```

Arguments

`...`	Arguments to be passed through the inherited constructors up to `refTable`.
`.model`	Numeric vector, to fill the `model` field of the object.
`warn`	Single logical value, to be passed to the `cghRA.regions` check method.

Value

An object of class `cghRA.regions`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.regions-class`, `track.table-class`, `refTable-class`

cghRA.series — *cghRA.series class constructor*

Description

This function returns a new cghRA.series object. Elements may be added to the series via the add method in a second time.

Usage

```
cghRA.series(..., .name, warn = TRUE)
```

Arguments

`...`	Elements to include in the series, as a single `list` or multiple variables containing cghRA.regions objects. Alternatively, a character vector of RDT file paths can be provided.
`.name`	Single character value, the name of the series.
`warn`	Single logical value, to be passed to the cghRA.series check method.

Value

An object of class cghRA.series.

Author(s)

Sylvain Mareschal

See Also

cghRA.series-class

cghRA.series-class — *Class* "cghRA.series"

Description

Objects from this class are collections of cghRA.regions objects, and provide various methods for CGH series analysis.

Objects from this class should always be produced by the cghRA.series constructor.

Extends

All reference classes extend and inherit methods from envRefClass.

Fields

`arrays`: A possibly named `list` of `cghRA.regions` objects.

`name`: Single character value, the custom name of the series.

Methods

`add(object)`: Add an object to the series

`applyMethod(.method, ..., .simplify = , .quiet = )`: Calls a method on each array of the series
- **.method** : single character value, the method to be called.
- **...** : arguments to be passed to the method.
- **.simplify** : same behavior as sapply() 'simplify' argument.
- **.quiet** : single logical value, whether to print iterations or not.

`check(warn = )`: Raises an error if the object is not valid, else returns TRUE

`get(arrayName)`: Returns an element from the series

`getArrayNames()`: Returns a vector of array names

`initialize(name = , arrays = , ...)`:

`last()`: Refers to the last array added in the series

`LRA(value = , tracks = , ...)`: Apply the LRA() function to list Long Recurrent Abnormalities (Lenz et al, PNAS 2008).
- **value** : single character value, the name of the column to use as copy number estimate ('copies' or 'logRatio').
- **tracks** : single logical value, whether to convert output to track.table class or not.

`parallelize(value = , quiet = , tracks = , ...)`: Apply the parallelize() function to build a summary matrix of the series.
- **tracks** : single logical value, whether to convert output to track.table class or not.

`penetrance(tracks = , ...)`: Apply the penetrance() function to compute the proportion of altered samples for each genomic position.
- **tracks** : single logical value, whether to convert output to track.table class or not.

`pool(tracks = , value = , group = , states = , others = , quiet = )`: Collect and pool all alterated segments from the various samples of the series.
- **tracks** : single logical value, whether to convert output to track.table class or not.
- **value** : column on which apply a filtering.
- **group** : single logical value, whether to visually group segments per samples or not (valid only for tracks=TRUE).
- **states** : list of states, see penetrance help page for details. If 'states' is not empty, segments without state will be filtered out.
- **others** : character vector, names of other columns to keep.
- **quiet** : single logical value, whether to throw diagnosis messages or not.

`SRA(value = , tracks = , ...)`: Apply the SRA() function to list Short Recurrent Abnormalities (Lenz et al, PNAS 2008).
- **value** : single character value, the name of the column to use as copy number estimate ('copies' or 'logRatio').
- **tracks** : single logical value, whether to convert output to track.table class or not.

`STEPS(tracks = , ...)`: Apply the STEPS() function to prioritize commonly altered regions.
- **tracks** : single logical value, whether to convert output to track.table class or not.

The following methods are inherited (from the corresponding class):

- callSuper (envRefClass)
- copy (envRefClass)
- export (envRefClass)
- field (envRefClass)
- getClass (envRefClass)
- getRefClass (envRefClass)
- import (envRefClass)
- initFields (envRefClass)
- show (envRefClass, overloaded)
- trace (envRefClass)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`cghRA.series`, `cghRA.regions`

`cnvScore` *Polymorphism likelihood score for a genomic segment*

Description

Computes for each genomic segment provided a score reflecting its likelihood to a polymorphism (CNV) dataset, as can be download from the Database of Genomic Variants.

Usage

```
cnvScore(sample.map, dgv.map, hangingThreshold = 0.8, minGain = 0.1, maxPaths = NA,
  trace = FALSE, expand = TRUE, quiet = TRUE)
```

Arguments

`sample.map`	A `segmentMap` object as returned by `map2design`, corresponding to the mapping of the segments to assess to a common design.
`dgv.map`	A `segmentMap` object as returned by `map2design`, corresponding to the mapping of the true polymorphism (CNV) dataset to a common design.
`hangingThreshold`	Single numeric value, segments to score must cover at least this proportion of union(CNV, segment) for a CNV to be considered. Decrease this value to allow poorly overlapping CNVs to (modestly) contribute to the final score, at the cost of longer computing time.
`minGain`	Single numeric value, CNVs must add at least this value to the path's score to be retained. Increase this value to allow poorly overlapping CNVs to (modestly) contribute to the final score, at the cost of longer computing time.
`maxPaths`	Single integer value, the maximal amount of paths to be computed for each segment (use `NA` to always compute all of them). Considering that most of the best paths are computed first and final score focus on them, an arbitrary value like 50 can be provided to decrease the computing time with marginal effects on the resulting scores.
`trace`	Single logical value, whether to produce a trace of every path constructed or only the final CNV score. This is mainly provided for debugging purpose, and increase the computing time. `trace2track` provides graphical means to visualize these traces.
`expand`	Single logical value, whether to return a vector of scores with one element for each row in `sample.map` (`FALSE`) or in the original mapped track (`TRUE`). As the mapping involves row compression (see `map2design`), producing a vector that can be directly used as a column in the original track needs an expansion step, that can be performed if requested via this argument.
`quiet`	Single logical value, whether to print diagnostic `message`s or not.

Value

If `trace` is `FALSE`, returns a numeric vector storing the resulting CNV score. See `expand` for further details on this vector size.

If `trace` is `TRUE`, returns a named `list` of two elements: "scores", that holds the numeric vector of scores (see above), and "traces", that described every path that has been built to compute the scores. The columns in "traces" are:

`seg`	Range of the original track indexes corresponding to the assessed segment.
`seg.score`	Final CNV score for the assessed segment, all paths comprised.
`path.count`	How many times the CNV path described was built.
`path.jaccard`	Jaccard index between the assessed segment and the CNV path described.
`path.cnvCount`	How many CNVs are included in the CNV path described.
`path.cnvList`	Indexes in `dgv.map` of the CNVs retained in the CNV path described.
`path.score`	'path.jaccard' corrected for the amount of CNVs included in the CNV path described.

Author(s)

Sylvain Mareschal

See Also

`map2design`, `applyMap`, `trace2track`

`copies` *LogRatio to copies conversion*

Description

`copies` applies a model to a vector of logRatios, converting them into copy amounts.

`LCN` is similar, but returns only Log-ratio related Copy Numbers, corresponding to a model with 0 as `center`, 1 as `width` and 2 as `ploidy`. See the references for further details on the models.

Usage

```
LCN(x, exact = TRUE)
copies(x, model = NA, center = model['center'], width = model['width'],
  ploidy = model['ploidy'], exact = TRUE, from = c("logRatios", "LCN", "copies"))
```

Arguments

`x`	Numeric vector, the values to be converted (their nature depends on `from`).
`model`	A numeric vector, as returned by `model.auto` or `model.test`. Can be `NA` if parameters are provided via other arguments.
`center`	Single numeric value, the most common `LCN` within the analyzed genome.
`width`	Single numeric value, `LCN` gaps between two consecutive real copy amounts.
`ploidy`	Single numeric value, the real copy amount corresponding to `center` `LCN`. A few altered human genome should have a ploidy of 2, use 0 to compute relative copy numbers rather than absolute ones.
`exact`	Single logical value, whether to `round` copy numbers or not.
`from`	Single character value defining what computation apply to `x`. "logRatios" assumes `x` to be logRatios to be converted to copy numbers, applying a full model (`center`, `width`, `ploidy`). "LCN" assumes `x` to be Log-ratio related Copy Numbers, as returned by `LCN`, so only the `exact` argument is used. "copies" assumes `x` to be already modelized copy numbers to be turned back into logRatios, using `ploidy` as reference.

Value

A numeric vector the same length as `x`.

Author(s)

Sylvain Mareschal

See Also

`model.auto`, `model.apply`

Examples

```
# Generating random segmentation results
## with 30% normal cells contamination
## with +10% for normal DNA labelling
segLogRatios <- c(
  rnorm(
    sample(5:20, 1),
    mean = log((1*0.7 + 2*0.3)/(2*1.1), 2),   # One deletion
    sd = 0.08
  ),
  rnorm(
    sample(80:120, 1),
    mean = log(2/(2*1.1), 2),                 # No alteration
    sd = 0.08
  ),
  rnorm(
    sample(40:60, 1),
    mean = log((3*0.7 + 2*0.3)/(2*1.1), 2),   # One more copy
    sd = 0.08
  )
)
segLogRatios <- sample(segLogRatios)
segLengths <- as.integer(3 + round(rchisq(length(segLogRatios), 1)*100))
segEnds <- cumsum(segLengths)
segStarts <- c(1L, head(segEnds, -1))
segChroms <- rep("chr1", length(segEnds))

# Generated genome
genome <- data.frame(
  segChroms,
  segStarts,
  segEnds,
  segLogRatios,
  segLengths
)
print(genome)

# Automatic modelization
model <- model.auto(
  segLogRatios = segLogRatios,
  segChroms = segChroms,
  segLengths = segLengths
)
```

```
# Relative copy numbers
print(
  copies(
    segLogRatios,
    model = model,
    ploidy = 0,
exact = FALSE
  )
)

# Absolute copy number (assuming n=2)
print(
  copies(
    segLogRatios,
    model = model,
    ploidy = 2,
    exact = FALSE
  )
)
```

`Design file parser` *Design file parser*

Description

These functions are examples of design file parsers, as can be used directly or by `tk.design` to produce a `cghRA.probes` object from a CGH design file.

Usage

```
Agilent.design(file, name = NULL, organism = as.character(NA),
  assembly = as.character(NA), chromosomes = NULL, ...)
custom.design(file, name = NULL, organism = as.character(NA),
  assembly = as.character(NA), chromosomes = NULL, ...)
```

Arguments

`file`	Single character value, path to the file to extract the design from (Agilent TDT design file for `Agilent.design`, CSV file as described below for `custom.design`).
`name`	Single character value, the name of the design. `NULL` will generate an automatic design name with the array dimensions (e.g. "Agilent 125 x 50").
`organism`	Single character value, the name of the organism studied by the current design.
`assembly`	Single character value, the genome assembly version for probe coordinates.
`chromosomes`	Character vector, the ordered list of the chromosome names for the design organism. If `NULL` the `factor` levels of the chrom column will be extracted, if not `chromosomes` will be used as levels to coerce the chrom column to `factor`.
`...`	Further arguments are ignored by `Agilent.design` and `custom.design`, but can be used by other design file parsers.

Details

As the package was developed with Agilent arrays, only the corresponding parser and a generic one are currently provided. Parsing design files from other brands can be achieved providing a custom design file parser suiting the manufacturer file format. Common brand file parsers may be added in the future, if you developed one (or need one to be developed) and wish it to be added to the package, please contact the package maintainer.

"Custom" files must be CSV files, using tabulations as column separators, periods as decimal separators and a first row naming columns. No comment line is allowed, and cell content protection (quoting) can be performed using double-quotes. The mandatory columns are "chrom" (character), "start" (integer) and "end" (integer), describing the genomic location of each probe in the design. Additionally it is recommended to provide "strand" ("+", "-" or `NA`), "id" (an integer ID that will be used to match probes between design and data files), "name" (character), "row" and "col" (integers, the physical position of the probe on the slide). Further columns will be stored as provided.

Value

An object of class `cghRA.design`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.design-class`, `tk.design`

`drawableFromClass.cghRA.probes`
Extend Rgb compatibility to cghRA.probes

Description

This function is only defined to allow the selection of RDT files containing `cghRA.probes` in Rgb `drawable.lists`. It should not be called directly by users.

Usage

```
drawableFromClass.cghRA.probes(track, design, ...)
```

Arguments

`track`	The `cghRA.probes` object extracted from the currently parsed RDT file.
`design`	Either a `cghRA.design` matching `track` or the path to a RDT file containing it. Alternatively a Tcl-tk dialog window will be summoned to select such a RDT file if `design` was not set in the `drawable.list$add()` call.
`...`	Further arguments are silently ignored.

Value

A `cghRA.array` object binding `track` and `design`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.array`

`fillGaps` *Fill gaps between consecutive segments*

Description

This function enlarges segments on their upper boundary to fill gaps between consecutive segments.

It may be crucial for `penetrance` computation, as they lead to small low steps in penetrance.

Usage

```
fillGaps(segTable, isOrdered = FALSE)
```

Arguments

`segTable`	A `data.frame` of segments, with at least "chrom" (character), "start" (integer) and "end" (integer) columns.
`isOrdered`	Single logical value, whether `segTable` is already ordered by chromosome and starting position or not.

Value

Returns a `data.frame` similar to `segTable`.

Author(s)

Sylvain Mareschal

GEDI — *Gene Expression and Dosage Integrator*

Description

This function implements the "Gene Expression and Dosage Integrator" CGH / transcriptome correlation, as described by Lenz et al.

Usage

```
GEDI(cgh, cgh.chrom, cgh.start, cgh.end, cgh.genes, expr, expr.genes,
  permutations = 1000, type = c("amplifications", "deletions"), quiet = FALSE)
```

Arguments

`cgh`	Logical matrix, with regions in rows and samples in columns. Alterated samples for a given region are to be `TRUE`, germline `FALSE` and other `NA`.
`cgh.chrom`	Character vector, the chromosome location of the regions described in `cgh`.
`cgh.start`	Integer vector, the starting position on the chromosome for the regions described in `cgh`.
`cgh.end`	Integer vector, the ending position on the chromosome for the regions described in `cgh`.
`cgh.genes`	Character vector, the names of the genes in each region described in `cgh`, separated by ", ". See the `cross` method of the `sliceable` class (in `Rgb` package) for an easy way to produce this, in combination with `track.NCBI_genes`.
`expr`	Numeric matrix of gene expressions, with probesets in rows and samples in columns.
`expr.genes`	Character vector, the names of the genes associated with each probeset described in `expr`, separated by ", ". Notice probesets associated with multiple genes will not be used, as they are not specific.
`permutations`	Single integer value, the amount of permutations to use for score computation. Time consumption and score accuracy increases with this value.
`type`	Single character value, describing the type of alterations studied (as the alternative hypothesis for the t-test depends on it).
`quiet`	Single logical value, when `FALSE` a message will be sent for each region processing, in order to evaluate the processing time.

Value

Returns a `list` with the following elements :

`gediScore`	Numeric vector with for each `cgh` row the proportion of permutated scores lesser than the observed one. The algorithm authors consider an association to be present if this score is greater than 0.9.

`gediGenes`	Character vector with for each `cgh` row the list of the genes used for the score computation (intersection of `cgh.genes` and `expr.genes` for the considered region).

Author(s)

Sylvain Mareschal

References

Lenz G et al. "Molecular subtypes of diffuse large B-cell lymphoma arise by distinct genetic pathways". Proc Natl Acad Sci U S A. 2008 Sep 9;105(36):13520-5 (Supporting Information)

`localize` *Localize CGH probes in a genome*

Description

`localize` returns genomic coordinates (chromosome, strand, starting position, ending position) of a set of probes into a given genome. It relies on the external `Blast-Like Alignment Tool` to perform fuzzy both-strands matching, and provides various filters suitable to CGH probes.

`blatInstall` needs to be executed once after the R package installation in order to use `localize`.

Usage

```
blatInstall(blat, cygwin)

localize(probeFile, chromFiles, chromPattern = "^(.+)\\.[^\\.]+$",
  blatArgs = character(0), rawOutput = FALSE, noMulti = TRUE, noOverlap = TRUE,
   noPartial = TRUE, verbose = 2)
```

Arguments

`blat`	Single character value, path to the BLAT executable file to use for localization.
`cygwin`	Single character value, path to the cygwin1.dll file that might be needed to run BLAT on Windows.
`probeFile`	Single character value, path to a multi-fasta file describing the probes to compute the bias for. FASTA comments are used as probe names, and should be unique.
`chromFiles`	Character vector, paths to chromosome sequences (a single fasta file for each chromosome).
`chromPattern`	Single character value, a regular expression to be used for chromosome name extraction from `chromFiles`. It needs to capture a single value for replacement, default value will use the base names of the files without extension as chromosome names.
`blatArgs`	Character vector, arguments to be passed to BLAT ("name=value" or "-flag"). See the BLAT documentation in 'References' for further details.

`rawOutput`	Single logical value, whether to return the merged BLAT output or the processed one (see 'Value'). Notice raw output is not filtered.
`noMulti`	Single logical value, whether to filter out probes located in multiple genomic positions or not. Ignored if `rawOutput`.
`noOverlap`	Single logical value, whether to filter out overlapping probes or not (when two overlapping probes are detected, both are discarded). Ignored if `rawOutput`.
`noPartial`	Single logical value, whether to filter out partial matches or not (they will still be used by other filters, to disable them completely consider using different BLAT arguments). Ignored if `rawOutput`.
`verbose`	Single numeric value, the level of verbosity (0, 1 or 2).

Value

If `rawOutput`, `localize` returns the tabular section of merged `psLayout 3` file returned by BLAT (see the BLAT documentation in 'References' for further details).

Else returns a `data.frame` with a row for each probe that was found and not filtered, ordered by `chrom`, `start` then `name` :

`name`	Character, the probe names, as defined by comments in `probeFile`.
`chrom`	Character, the chromosomal location of the probe, as defined by the `chromNames` corresponding to the codechromFiles in which the probe matched.
`strand`	Character, "+" for a forward match, "-" for a reverse complement match.
`start`	Integer, the lower position of the probe in the chromosome. See 'Coordinate system'.
`end`	Integer, the upper position of the probe in the chromosome. See 'Coordinate system'.
`insertions`	Integer, amount of nucleotides inserted in the probe when refering to the chromosome sequence.
`deletions`	Integer, amount of nucleotides deleted in the probe when refering to the chromosome sequence.
`mismatches`	Integer, amount of mismatching nucleotides between probe and chromosome sequence.
`freeEnds`	Integer, amount of nucleotides at probe extremities ignored in the alignment.

Coordinate system

When `rawOutput` is `FALSE`, coordinates begin at 1, both boundaries are comprised in the sequence and length can be computed as `end - start + 1` (`Biostrings` behavior).

When `rawOutput`, refer to BLAT specifications (See 'References').

In both cases, backward matches (strand = "-") are expressed in forward coordinates (start < end) (BLAT behavior).

BLAT installation

BLAT relies on a single executable file, so installation is straight-forward.

Download the executable file or compile it for your computer architecture, then simply use the `blatInstall` function to copy it to the proper package folder for further uses. Precompiled executables for various systems can be found on the author website (see 'References'), as part of the BlatSuite (only 'blat.exe' or 'blat' is needed).

Windows specificities: Running BLAT on Windows needs Cygwin. You can install Cygwin entirely on your system (see 'References'), or download the "cygwin1.dll" file and provide it to `blatInstall`, as it is the only Cygwin component needed. DLL is a common format for informatic viruses, so be sure of the website you download this file from. You can safely (no guarantee !) download it from the official website (see 'References') mirrors, they generally keep compressed archives in /release/cygwin in which you can find the DLL (in /usr/bin).

Author(s)

Sylvain Mareschal

References

BLAT is an open-source software freely available for academic, nonprofit and personal use. See the FAQ for further details. FAQ, specifications, source code and executables

Cygwin is a free and open-source software under GNU General Public Licencing. Official website

See Also

`bias`

`map2design` *Update a track coordinates to match a distinct CGH design*

Description

Remapping a `track.table` object storing genomic segments to a specific CGH design consists of two steps :

- The production of a map, which defines the coordinates of each segment by the indexes of the first and last CGH probes included in it (`map2design`).
- The update of the genomic coordinates of the original track, using the map and the design (`applyMap`).

Usage

```
map2design(track, design, minProbes = 1, quiet = FALSE, warn = TRUE)
applyMap(track, map, design)
```

Arguments

`track`	A `track.table`-inheriting object, storing one row for each genomic segment of interest in a CGH-like experiment.
`design`	A `track.table`-inheriting object (preferably a `cghRA.design` object), storing one row for each probe in the design data is to be remapped on.
`minProbes`	Single integer value, the amount of probes a segment in `track` must cover to be retained.
`quiet`	Single logical value, whether to print diagnostic `message`s or not.
`map`	An integer matrix defining the mapping of `track` to `design`, as produced by `map2design`.
`warn`	Single logical value, to be passed to the check method of the newly created `segmentMap` object.

Value

`map2design` returns an integer matrix with 3 columns and row names. Columns "start" and "end" define the coordinates of a segment as probe indexes in `design`, and column "count" allow to group segments with the same remapped coordinates. Row names correspond to the index range of the corresponding segments in the original `track`.

`applyMap` returns a copy of `track`, in which `start` and end coordinates have been updated to match coordinates of probes in `design`. Segments that do not overlap at least `minProbes` probe in `design` are excluded.

Author(s)

Sylvain Mareschal

See Also

`cnvScore`

`model.apply` *Computes copy number for a set of CGH segments*

Description

This function translates log ratios of a set of segments into copy numbers, applying a copy number model as produced by `model.auto` or `model.test`.

If `exact` is set set to `FALSE`, copy numbers are rounded and consecutive segments with the same copy number are merged.

Usage

```
model.apply(segStarts, segEnds, segChroms, segLogRatios, segLengths, model = NA,
  center = model['center'], width = model['width'], ploidy = model['ploidy'],
  exact = FALSE, merge = TRUE)
```

Arguments

`segStarts`	Numeric vector, the starting positions of the CGH segments to modelize.
`segEnds`	Numeric vector, the endind positions of the CGH segments to modelize.
`segChroms`	Vector, the chromosome holding the CGH segments to modelize.
`segLogRatios`	Double vector, the log ratios of the CGH segments to modelize.
`segLengths`	Numeric vector, the lengths of the CGH segments to modelize.
`model`	A numeric vector, as returned by `model.auto` or `model.test`. Can be `NA` if parameters are provided via other arguments.
`center`	Single double value, the center parameter to use in the model.
`width`	Single double value, the width parameter to use in the model.
`ploidy`	Single numeric value, copy number supposed to be the most common within the analyzed genome.
`exact`	Single logical value, whether to return continue copy numbers (double) or discrete ones (integer).
`merge`	Single logical value, whether to merge consecutive segments with the same copy number when `exact` is `FALSE`.

Value

Returns a `data.frame` describing the segments :

`segStarts`	Extracted from the `segStarts` argument.
`segEnds`	Extracted from the `segEnds` argument.
`segChroms`	Extracted from the `segChroms` argument.
`segLogRatios`	Double, the theoretic log ratio of the segment, with 2 copies as reference.
`segCopies`	Numeric, the copy number of the segment.
`segLengths`	Extracted from the `segLengths` argument.

Author(s)

Sylvain Mareschal

See Also

`copies`, `model.auto`, `model.test`

Examples

```
# Generating random segmentation results
## with 30% normal cells contamination
## with +10% for normal DNA labelling
segLogRatios <- c(
  rnorm(
    sample(5:20, 1),
    mean = log((1*0.7 + 2*0.3)/(2*1.1), 2),   # One deletion
    sd = 0.08
  ),
  rnorm(
    sample(80:120, 1),
    mean = log(2/(2*1.1), 2),                 # No alteration
    sd = 0.08
  ),
  rnorm(
    sample(40:60, 1),
    mean = log((3*0.7 + 2*0.3)/(2*1.1), 2),   # One more copy
    sd = 0.08
  )
)
segLogRatios <- sample(segLogRatios)
segLengths <- as.integer(3 + round(rchisq(length(segLogRatios), 1)*100))
segEnds <- cumsum(segLengths)
segStarts <- c(1L, head(segEnds, -1))
segChroms <- rep("chr1", length(segEnds))

# Generated genome
genome <- data.frame(
  segChroms,
  segStarts,
  segEnds,
  segLogRatios,
  segLengths
)
print(genome)

# Automatic modelization
model <- model.auto(
  segLogRatios = segLogRatios,
  segChroms = segChroms,
  segLengths = segLengths
)

# Profile simplification
segments <- model.apply(
  segStarts,
  segEnds,
  segChroms,
  segLogRatios,
  segLengths,
  model = model,
  exact = FALSE,
  merge = TRUE
)
layout(matrix(1:2, ncol=1))
plot(x=segStarts, y=segLogRatios, type="s", xlab="Position", ylab="Log Ratios")
plot(x=segments$segStarts, y=segments$segCopies, type="s", xlab="Position", ylab="Copies")
print(segments)

layout(1)
```

`model.auto` *Automatic generation of copy number model*

Description

This function computes a copy number model, as needed by `model.apply` to translate logRatios into copy numbers.

Usage

```
model.auto(segLogRatios, segChroms, segLengths = rep(1, length(segLogRatios)),
  from = 0.02, to = 0.5, by = 0.001, precision = 512, maxPeaks = 8, minWidth = 0.15,
   maxWidth = 0.9, minDensity = 0.001, peakFrom = -2, peakTo = 1.3, ploidy = 0,
  discreet = FALSE, method = c("stm", "sdd", "ptm"), exclude = c("X", "Y", "Xp", "Xq",
   "Yp", "Yq"))
```

Arguments

`segLogRatios`	Double vector, the log ratios of the CGH segments to modelize.
`segChroms`	Vector, the chromosome holding the CGH segments to modelize.
`segLengths`	Double vector, the lengths of the CGH segments to modelize. Amount of probes should be prefered if available, but nucleotide length or no length at all can also be used.
`from`	Single double value, the minimal bandwidth to test for `density`.
`to`	Single double value, the maximal bandwidth to test for `density`.
`by`	Single double value, the precision of the bandwidths to test for `density`.
`precision`	Single integer value, the amount of points to compute for `density`. As its help page suggests, values greater than 512 should be powers of 2.
`maxPeaks`	Single integer value, the maximal amount of peaks in the density of distribution to consider a model as valid.
`minWidth`	Single double value, minimal value allowed for the `width` model parameter (thus for tumoral cell proportion in the sample).
`maxWidth`	Single double value, maximal value allowed for the `width` model parameter (thus for tumoral cell proportion in the sample).
`minDensity`	Single double value, minimal density for a peak to be detected.
`peakFrom`	Single double value, minimal logRatio for a peak to be detected. Use `NA` for no lower limit. Only 1, 2 and 3 copies peaks should be considered for a more precise model.
`peakTo`	Single double value, maximal logRatio for a peak to be detected. Use `NA` for no upper limit. Only 1, 2 and 3 copies peaks should be considered for a more precise model.
`ploidy`	Single numeric value, copy number supposed to be the most common within the analyzed genome.

`discreet`	Single logical value, if `FALSE` a fail in modelization raises an error, if `TRUE` it returns a `NA` filled model.
`method`	Single character value, the statistic to minimize ("stm" is default). See below for further details.
`exclude`	Vector, the chromosomes to exclude from the density computation and to plot with distinct symbols (use `NULL` to disable this feature). Sexual chromosomes should be excluded in heterogeneous DNA source, as their desequilibrium (2 'X' and no 'Y' in women) impact normal cells AND tumoral ones.

Details

More details about the cghRA copy number model and modelization can be found in the vignette associated with this package, as well as in the related publication. Once the parameters of a model (width and center) are set, three scores can be computed to assess its fitness to the data :

STM is the "Segment To Model" score, computed at the segment level as the average of the residuals weighted by the segment size (in probe counts). Residuals are computed as the absolute difference between exact copy numbers (see the `copies` function) and their rounding, assuming that copy numbers should be integers and that decimal parts are noise in the model. This is the recommended score to use with cghRA.

PTM is the "Peak To Model" score, computed at the peak level as the average of the residuals. Residuals are computed as the absolute difference between exact copy numbers (see the `copies` function) and their rounding, assuming that copy numbers should be integers and that decimal parts are noise in the model.

SDD is the "Standard Deviation of peak Differences" score. As its name suggests, it is computed as the `sd` or differences between consecutive peaks, considering that good models should show very regularly spaced density peaks.

Value

Returns a double vector, with the following values :

`bw`	Bandwidth used for `density` computation.
`peaks`	Amount of peaks considered in the model.
`peakFrom`	See the `peakFrom` argument.
`peakTo`	See the `peakTo` argument.
`center`	Center parameter of the model.
`width`	Width paremeter of the model.
`ploidy`	Ploidy paremeter of the model, as provided.
`sdd`	Quality statistic, see 'Details'.
`ptm`	Quality statistic, see 'Details'.
`stm`	Quality statistic, see 'Details'.

Author(s)

Sylvain Mareschal

See Also

model.test, model.apply

Examples

```
# Generating random segmentation results
## with 30% normal cells contamination
## with +10% for normal DNA labelling
segLogRatios <- c(
  rnorm(
    sample(5:20, 1),
    mean = log((1*0.7 + 2*0.3)/(2*1.1), 2),   # One deletion
    sd = 0.08
  ),
  rnorm(
    sample(80:120, 1),
    mean = log(2/(2*1.1), 2),                 # No alteration
    sd = 0.08
  ),
  rnorm(
    sample(40:60, 1),
    mean = log((3*0.7 + 2*0.3)/(2*1.1), 2),   # One more copy
    sd = 0.08
  )
)
segLogRatios <- sample(segLogRatios)
segLengths <- as.integer(3 + round(rchisq(length(segLogRatios), 1)*100))
segEnds <- cumsum(segLengths)
segStarts <- c(1L, head(segEnds, -1))
segChroms <- rep("chr1", length(segEnds))

# Generated genome
genome <- data.frame(
  segChroms,
  segStarts,
  segEnds,
  segLogRatios,
  segLengths
)
print(genome)

# Automatic modelization
model <- model.auto(
  segLogRatios = segLogRatios,
  segChroms = segChroms,
  segLengths = segLengths
)
print(model)
```

model.test *Copy number model quality assessment*

Description

This function provides various data to manually fit or upgrade a copy number model, as needed by `model.apply` to translate logRatios into copy numbers.

Usage

```
model.test(segLogRatios, segChroms, segLengths = rep(1, length(segLogRatios)),
   model = NA, center = model['center'], width = model['width'],
   ploidy = model['ploidy'],  bw = model['bw'], minDensity = 0.001,
   peakFrom = model['peakFrom'], peakTo = model['peakTo'], graph = TRUE,
  parameters = TRUE, returnPar = FALSE, xlim = c(0, 5), ylim = c(0, max(segLengths)),
  xlab = "Segment copy number", ylab = "Segment length", cex.seg = 0.4, cex.leg = 0.7,
   cex.l2r = 0.7, exclude = c("X", "Y", "Xp", "Xq", "Yp", "Yq"), title = NULL,
   panel = FALSE, klim = NULL, ...)
```

Arguments

`segLogRatios`	Double vector, the log ratios of the CGH segments to modelize.
`segChroms`	Vector, the chromosome holding the CGH segments to modelize.
`segLengths`	Double vector, the lengths of the CGH segments to modelize. Amount of probes should be prefered if available, but nucleotide length or no length at all can also be used.
`model`	A double vector, as returned by `model.auto` or `model.test`. Can be `NA` if parameters are provided via other arguments.
`center`	Single double value, the center parameter to use in the model.
`width`	Single double value, the width parameter to use in the model.
`ploidy`	Single numeric value, copy number supposed to be the most common within the analyzed genome.
`bw`	Single double value, the bandwidth parameter to use in the model.
`minDensity`	Single double value, minimal density for a peak to be detected.
`peakFrom`	Single double value, the peak logRatio lower limit parameter to use in the model.
`peakTo`	Single double value, the peak logRatio upper limit parameter to use in the model.
`graph`	Single logical value, whether to plot the density distribution of the segments with the modelized copy numbers or not.
`parameters`	Single logical value, whether to add a legend to the plot with the parameters and statistics of the model or not.
`returnPar`	Single logical value, whether to return the `par` content (for point identification in interactive plots) or the model statistics.

`xlim`	Vector of two double values, the boundaries of the plot on the horizontal axis (in `LCN`).
`ylim`	Vector of two double values, the boundaries of the plot on the vertical axis (in the same units than `segLengths`).
`xlab`	Single character value, the title to print for the horizontal axis.
`ylab`	Single character value, the title to print for the vertical axis.
`cex.seg`	Single double value, the character expansion factor for points (segments) on the plot.
`cex.leg`	Single double value, the character expansion factor for the plot legend.
`cex.l2r`	Single double value, the character expansion factor for the log-ratio axis of the plot.
`exclude`	Vector, the chromosomes to exclude from the density computation and to plot with distinct symbols (use `NULL` to disable this feature). Sexual chromosomes should be excluded in heterogeneous DNA source, as their desequilibrium (2 'X' and no 'Y' in women) impact normal cells AND tumoral ones.
`title`	To be passed to `legend`, see there for allowed types (usually a single character value).
`panel`	Single logical value, whether to plot a rotated minimalist graph or a classic one.
`klim`	Double vector of two values, alternative definition of `xlim` in modelized copy numbers rather than LCN.
`...`	Further graphical arguments to be passed to `plot`.

Value

When `returnPar` is `TRUE`, invisibly returns the `par` content, for point identification.

When `returnPar` is `FALSE`, returns the same vector as `model.auto`, see its help page for further details.

Author(s)

Sylvain Mareschal

See Also

`model.auto`, `model.apply`

Examples

```
# Generating random segmentation results
## with 30% normal cells contamination
## with +10% for normal DNA labelling
segLogRatios <- c(
  rnorm(
    sample(5:20, 1),
    mean = log((1*0.7 + 2*0.3)/(2*1.1), 2),   # One deletion
    sd = 0.08
    ),
    rnorm(
      sample(80:120, 1),
      mean = log(2/(2*1.1), 2),                 # No alteration
      sd = 0.08
    ),
    rnorm(
      sample(40:60, 1),
      mean = log((3*0.7 + 2*0.3)/(2*1.1), 2),   # One more copy
      sd = 0.08
    )
  )
  segLogRatios <- sample(segLogRatios)
  segLengths <- as.integer(3 + round(rchisq(length(segLogRatios), 1)*100))
  segEnds <- cumsum(segLengths)
  segStarts <- c(1L, head(segEnds, -1))
  segChroms <- rep("chr1", length(segEnds))

  # Generated genome
  genome <- data.frame(
    segChroms,
    segStarts,
    segEnds,
    segLogRatios,
    segLengths
  )
  print(genome)

  # Automatic modelization
  autoModel <- model.auto(
    segLogRatios = segLogRatios,
    segChroms = segChroms,
    segLengths = segLengths
  )

  layout(matrix(1:2, ncol=1))

  # Show automatic model
  model.test(
    segLogRatios = segLogRatios,
    segChroms = segChroms,
    segLengths = segLengths,
model = autoModel
  )

  # Standard model derived from the log ratios definition
  refModel <- model.test(
    segLogRatios = segLogRatios,
    segChroms = segChroms,
    segLengths = segLengths,
    center = 2,
    width = 1,
    bw = 0.1        # Arbitrary
```

```
)

# Differences in scores
print(autoModel)
print(refModel)

layout(1)
```

`parallelize` *Reshapes a list of segments*

Description

This function reshapes a list of segment `data.frames` (with chromosomal location and value) into a single `data.frame` containing a column for each element of the list (typically samples) and a the minimal amount of regions in rows.

Usage

```
parallelize(segTables, value = "logRatio", digits = 3, quiet = FALSE, chroms = NULL)
```

Arguments

`segTables`	An eventually named `list` of `data.frames` to reshape. All the `data.frames` must contain at least "chrom" (character), "start" (integer), "end" (integer) columns, and the column defined by `value`. Can also be a single `data.frame` containing all the segments, with a `.sampleIdentity` integer column.
`value`	Single character value, the column name from which extract values that will fill the output cells.
`digits`	Single integer value to be passed to `round` for each cell of the output (`NA` disables the rounding step).
`quiet`	Single logical value, whether to throw diagnosis messages or not.
`chroms`	Character vector, the names of chromosomes to restrain the analysis on (frequently autosomes). If `NULL`, all chromosomes in `segTable` will be used.

Value

Returns a `data.frame` with the following columns :

`chrom`	Character, the chromosomal location of the region described.
`start`	Integer, the lower coordinate of the region described.
`end`	Integer, the upper coordinate of the region described.
`...`	For each element of `segTables` a column with the value extracted from the `value` column of the according `data.frame`.

Author(s)

Sylvain Mareschal

See Also

penetrance

`parseKaryo` *Parses a karyotype-like formula*

Description

This function produces a `cghRA.regions` object from a simplified karyotype formula, associating copy numbers to numeric coordinates.

Usage

```
parseKaryo(formula, bandTrack, name = as.character(NA), design = NULL,
  alteratedOnly = TRUE)
```

Arguments

`formula`	Single character value, the formula to be parsed. See 'Examples'.
`bandTrack`	A `track.table` object with cytoband definition, as returned by the `track.UCSC_bands` function from the `Rgb` package.
`name`	Single character value, to be used as name for the produced object.
`design`	A `cghRA.design` object, or `NULL`. If provided, a `cghRA.copies` object will be produced, using `design` to compute probe content of each region. Else, a `track.table` object will be returned.
`alteratedOnly`	Single logical value, if `TRUE` normal clones (2n without alteration) will not be averaged with alterated clones for the final copy amount computation. If all clones are normals, a normal genome will be returned anyway.

Value

Returns a `list` with two elements : "clones" and "copies".

"clones" is a summary of the clones found in the formula as an integer value, with mitosis counts as values and ploidy as names.

"copies" is a `track.table`-inheriting object with genomic regions of distinct copy numbers. If `design` is provided, the object is a `cghRA.copies` object, else a `track.table` object.

Author(s)

Sylvain Mareschal

See Also

cghRA.copies

Examples

```
  ## Not run:
    karyo <- paste(
      "111<5n>,6(1qt-p11),4(1p11-pt),4(2),8(3),4(4),6(5),6(6pt-q22),6(6q26-qt),",
      "2(6q22-q26),6(7pt-q31),3(7q31-qt),6(9),4(10),4(11),4(12),6(13),4(14),",
      "4(15pt-q22),2(15q22-qt),2(16),4(17),6(18),4(19),4(21),4(22) [6] ; 46<2n> [7]",
      collapse = ""
    )
    parseKaryo(karyo, bandTrack)

## End(Not run)
```

`penetrance` *Penetrance computation from a series of segments*

Description

This function computes the penetrance of various states from a parallelized series of segments.

In each point of the genome, the penetrance is the proportion of the series arrays that show a specific alteration state.

Usage

```
penetrance(segParallel, states = list(deletion=c(-Inf, -0.5), gain=c(0.5, Inf)),
  na = c("fill", "keep", "false"), mergeOnValue = FALSE, bool = FALSE, quiet = FALSE)
```

Arguments

`segParallel`	A `data.frame`, as returned by parallelize.
`states`	A named `list` of numerics defining the boundaries of each state. Each state may be defined by a single value (the only value in `segParallel` to link to the state) or by two boundaries (the lower boundary is part of the state, the upper one is not). `Inf` and `-Inf` can be used as boundaries.
`na`	Single character value defining how to deal with `NA` segments : "fill" fills them when possible (chromosome ends and gaps for which the state is the same on each side), "keep" keeps all of them `NA` and "false" always considers them as "not in the state". When `NA` remains ("fill" or "keep"), the penetrance frequency is locally computed on non-`NA` samples.
`mergeOnValue`	Single logical value, whether to merge consecutive regions with same penetrance value but distinct alterated sample list.
`bool`	Single logical value, if `TRUE` the penetrance is not returned but logical matrixes of regions 'in state' are returned instead. This is a quite uncommon behavior, allowed essentially for code recycling by other packages, use `FALSE`.
`quiet`	Single logical value, whether to throw diagnosis messages or not.

Value

If bool is FALSE, a list containing a distinct data.frame for each state, with the following columns :

chrom	Character, the chromosomal location of the region described.
start	Integer, the lower coordinate of the region described.
end	Integer, the upper coordinate of the region described.
value	Numeric, the penetrance in the region described for the state described.

Author(s)

Sylvain Mareschal

See Also

parallelize, STEPS

Probe file parser *Probe file parser*

Description

These functions are examples of probe file parsers, as requested by process to produce a cghRA.probes object from a CGH array data file.

Usage

```
Agilent.probes(
  file,
  columns = c(
    rFin = "rProcessedSignal",
    gFin = "gProcessedSignal",
    flag_rIsSaturated = "rIsSaturated",
    flag_gIsSaturated = "gIsSaturated",
    flag_rIsFeatNonUnifOL = "rIsFeatNonUnifOL",
    flag_gIsFeatNonUnifOL = "gIsFeatNonUnifOL",
    flag_rIsBGNonUnifOL = "rIsBGNonUnifOL",
    flag_gIsBGNonUnifOL = "gIsBGNonUnifOL",
    flag_rIsFeatPopnOL = "rIsFeatPopnOL",
    flag_gIsFeatPopnOL = "gIsFeatPopnOL",
    flag_rIsBGPopnOL = "rIsBGPopnOL",
    flag_gIsBGPopnOL = "gIsBGPopnOL"
  ),
  ...
)
custom.probes(file, columns = NULL, ...)
```

Arguments

`file`	Single character value, path to the file to extract the design from (Agilent Feature Extraction file).
`columns`	Character vector defining the columns to extract, the names are the names to use in the `cghRA.probes` object while the values are the names used in the Feature Extraction file.
`...`	Further arguments are ignored by `Agilent.probes` and `custom.probes`, but can be used by other probe file parsers.

Details

As the package was developped with Agilent arrays, only the corresponding parser and a generic one are currently provided. Parsing arrays from other brands can be achieved providing a custom probe file parser suiting the manufacturer file format. Common brand file parsers may be added in the future, if you developped one (or need one to be developped) and wish it to be added to the package, please contact the package maintainer.

As this function will be exported for parallel computing, dependencies need to be explicit : packages need `library` calls (even the core ones) or usage of `::` operators and sub-functions should be declared inside the parser body.

"Custom" files must be CSV files, using tabulations as column separators, periods as decimal separators and a first row naming columns. No comment line is allowed, and cell content protection (quoting) can be performed using double-quotes. The mandatory columns are "id" (an integer ID that will be used to match probes between design and data files) and "logRatio" (numeric). Additionally one can provide boolean columns starting with "flag_", to be used as probe filters by `process.mask` during the array processing. Further columns will be stored as provided.

Value

An object of class `cghRA.probes`.

Author(s)

Sylvain Mareschal

See Also

`cghRA.probes-class`

`process` *cghRA array processing*

Description

These functions implement the cghRA workflow, as a sequence of `process` subfunction calls. Each of them rely on `cghRA.array` and `cghRA.regions` methods, so custom processing can be easily achieved using them directly if the `steps` argument is not flexible enough to your purpose.

Custom steps can be added as well on the model of existing ones, defining a function called `process.NAME` and adding "NAME" to the `steps` vector during the call to `process`. Step functions need to handle at least an `input` parameter which will be returned directly by the previous step, thus forming a pipeline.

The `tk.process` function is a wrapper for `process`, built around a Tcl-Tk interface for more user-friendliness.

The `process` function is a multi-core command line interface that will dispatch its arguments to individual `process.core` calls, and should be the prefered entry point even on single core computers. `process.log` is a wrapper to `process.core` which captures warnings and errors into a log file.

The `process.default` function is a common way for `process` and `tk.process` to obtain default values for complex arguments like 'segmentArgs' and 'modelizeArgs'. It can be used to obtain the profiles proposed by `tk.process` in `process`.

Usage

```
process(inputs, logFile = "process.log", cluster = NA, ...)
process.log(..., logFile)
process.core(input, inputName, steps = c("parse", "mask", "replicates", "waca",
  "export", "spatial", "segment", "fill", "modelize", "export", "fittest", "export",
  "applyModel", "export"), ...)
process.parse(input, design, probeParser = Agilent.probes, probeArgs = list(), ...)
process.probes(input, design, ...)
process.regions(input, ...)
process.mask(input, ...)
process.replicates(input, replicateFun = stats::median, ...)
process.waca(input, ...)
process.spatial(input, outDirectory, ...)
process.segment(input, segmentArgs = process.default("segmentArgs"), ...)
process.fill(input, ...)
process.modelize(input, modelizeArgs = process.default("modelizeArgs"), ...)
process.applyModel(input, ...)
process.fittest(input, ...)
process.export(input, outDirectory, ...)
tk.process(globalTopLevel, localTopLevel)
process.default(argName, profileName)
```

Arguments

`inputs`	List of `input` to dispatch to each node (preferably named). The default workflow expects it to be a character vector naming raw data files to be parsed.
`logFile`	Single character value, the path to the log file to produce with messages, warnings and errors. If the file already exists, it will be emptied first. The behavior

	when `logFile` is set to `NA` or "" depends on `cluster`: if `cluster` is `FALSE` (unparallelized mode), messages and errors will be passed to the R console rather than logged in a file, if `cluster` is anything else they will be silently ignored.
`cluster`	Arguments to be passed to `makeCluster` as a list, for parallel processing (requires the optionnal `parallel` package). Remote machines are not handled properly in the current version of `process`, you should limit to "spec" defining how many processors can be used on the local machine as an integer value. The `FALSE` value requires an unparallelized mode, slower but more suitable for error tracking. The `NA` default value tries to detect the CPU count on the local machine if `parallel` is installed, else switches to unparallelized mode.
`...`	Further arguments to be passed to `process` sub-functions, depending on the `steps` choosen (see below). The default workflow expects at least `design` and `outDirectory` to be provided.
`input`	A single input to process on one node. The default workflow expects it to be a single character value naming a raw data file to be parsed.
`inputName`	Single character value, the name of the input currently processed (for logging only).
`steps`	Ordered character vector, naming the processing steps to apply. Custom steps can be named as well, as long as a function named "process.[step]" exists in the global environment. Each step will take as input the output of the previous step, the first step taking the value of the `input` argument as input.
`probeParser`	The function to parse `probeFiles` into `cghRA.probes` objects, such as `Agilent.probes` for Agilent FeatureExtraction arrays.
`probeArgs`	A list of arguments to pass to `probeParser` (apart from 'file' which is always provided).
`design`	Single character vector, the path and name of the RDT design file, as produced by `tk.design`.
`replicateFun`	The function to apply to replicate groups, if the "replicate" step is to be applied. This function must use a vector of numeric values (logRatios) as input, and return a single representative value (typically `median` or `mean`).
`outDirectory`	Single character value, the directory in which produce the output files.
`segmentArgs`	Character vector, the arguments to be passed to the `DNAcopy` method of the `cghRA.array` class. Arguments are defined as a character string that will be parsed, multiple values define multiple segmentation profiles to apply sequentially.
`modelizeArgs`	Single character value, the arguments to be passed to the `model.auto` method of the `cghRA.array` class. Arguments are defined as a character string that will be parsed.
`argName`	Single character value, 'segmentArgs' or 'modelizeArgs', the argument to get the default value for. If missing, the list of profiles and arguments handled is returned.
`profileName`	Single character value, altering the default values returned. If missing, the default profile is returned.

`globalTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the top level of the embedding interface, generally a call to `tktoplevel`.
`localTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the local top level to use to build this interface, generally a `tkframe` or `ttkframe`.

Value

Only `process.default` returns something : if `argName` is provided it returns the default value for the queried argument, else a list of profiles available for each handled argument. When many profiles are handled, the first value in the list is the default one (returned when `profileName` is missing).

Processing steps

The complete workflow involves the following steps :

parse Read a raw data file and return a `cghRA.array` object.

probes Read a `cghRA.probes` object stored in a RDT file and return a `cghRA.array` object.

regions Reads one or many `cghRA.regions` file(s) stored in RDT file(s).

mask Discard flagged probes (saturated, high background ...) in a `cghRA.array` object. Any `TRUE` value in a column whose name begins with "flag_" is enough to discard a probe (turn its logRatio into `NA`. See the cghRA.array$maskByFlag() method for further details.

replicates Replace replicated probe groups (same "name") by a single representative value (all logRatios are turned to `NA` except from the first one which will hold the representative value). See the cghRA.array$replicates() method for further details.

waca Apply the WACA algorithm to the logRatios. See the cghRA.array$WACA() method for further details.

spatial Produce a PNG file to visually check spatial biases. See the cghRA.array$spatial() method for further details.

segment Compute regions with similar logRatios along the genome, using the CBS algorithm. See the cghRA.array$DNAcopy() method for further details.

fill Extend segments to the right to join consecutive segments. See the cghRA.regions$fillGaps() method for further details.

modelize Fit a copy number model to segments, in order to convert logRatios to true copy numbers. If `segmentArgs` contains multiple values, each segmentation profile will lead to distinct "copies" and "regions" files numbered according to its position in `segmentArgs`. See the cghRA.regions$model.auto() method for further details.

applyModel Convert a modelized `cghRA.regions` objects into `cghRA.copies`.

fittest If multiple segmentation profiles have been used, select the fittest model ("copies" and "regions" files duplicated without number). For further details on the STM score used for fittest model selection, see the `model.auto` function of the cghRA.copies package.

clean Erase "copies" and "regions" files of the different segmentation profiles tested, as "fittest" should have saved the best.

Author(s)

Sylvain Mareschal

See Also

`tk.design`, `cghRA.array`

`segmentMap-class` *Class* `"segmentMap"`

Description

Efficient storage of a large collection of genomic intervals, located using probe IDs from a specific array design rather than genomic coordinates. Objects of this class are essentially intended to be produced by the `map2design` function, and used by the `cnvScore` function.

Extends

All reference classes extend and inherit methods from `envRefClass`.

Fields

`designName`**:** Single character value, the content of the `name` field of the `cghRA.design` object used to produce the object.

`designSize`**:** Single integer value, the row count in the `cghRA.design` object used to produce the object.

`map`**:** Integer matrix with one row for each distinct genomic interval in the mapped `track.table` object. The columns are `start` and `end`, the indexes of the first and last design elements in the interval and `count`, the amount of such intervals in the mapped object. Row names of this matrix list the indexes of the corresponding mapped object intervals.

`trackName`**:** Single character value, the content of the `name` field of the mapped `track.table` object.

`trackSize`**:** Single integer value, the row count in the mapped `track.table` object.

Methods

`check(warn = )`**:** Raises an error if the object is not valid, else returns TRUE

`initialize(map = , trackName = , trackSize = , designName = , designSize = , ...)`**:**

The following methods are inherited (from the corresponding class):

- callSuper (envRefClass)
- copy (envRefClass)
- export (envRefClass)
- field (envRefClass)

- getClass (envRefClass)
- getRefClass (envRefClass)
- import (envRefClass)
- initFields (envRefClass)
- show (envRefClass, overloaded)
- trace (envRefClass)
- untrace (envRefClass)
- usingMethods (envRefClass)

Author(s)

Sylvain Mareschal

See Also

`map2design`, `cnvScore`

`SRA & LRA` *Short/Long Recurrent Abnormalities detection*

Description

These functions extract Short Reccurent Abnormalities (SRA) and Long Reccurent Abnormalities (LRA) from a CGH array series, as described by Lenz et al. (2008).

The processing core `xRA` is common for both analysis, but is not intended to be called directly. Use the `SRA` and `LRA` wrappers instead.

Usage

```
xRA(segTables, value = "copies", states = list(deletion=c(-Inf,-0.5), gain=c(0.5,Inf)),
  sampleMin = 2, quiet = FALSE, lengthMax, lengthMin, gaps.width, gaps.ratio)
SRA(...)
LRA(...)
```

Arguments

`segTables`	An eventually named `list` of `data.frames` to reshape. All the `data.frames` must contain at least "chrom" (character), "start" (integer), "end" (integer) columns, and the column defined by `value`. Can also be a single `data.frame` containing all the segments, with a `.sampleIdentity` integer column.
`value`	Single character value, the column name from which extract values that will fill the output cells.

`states`	A named `list` of numerics defining the boundaries of each state. Each state may be defined by a single value (the only value in `segParallel` to link to the state) or by two boundaries (the lower boundary is part of the state, the upper one is not). `Inf` and `-Inf` can be used as boundaries.
`sampleMin`	Single numeric value, minimal amount of samples in the 'overlapping group'. If lesser than 1, interpreted as a proportion of the sample count. Large values decrease processing time and SRA amounts.
`quiet`	Single logical value, whether to print diagnostic `message`s or not.
`lengthMax`	Single integer value, segments larger than this value will be filtered out (25 Mb for SRA, `NA` for LRA). Use `NA` to disabled length filtering.
`lengthMin`	Single integer value, segments shorter than this value will be filtered out (`NA` for SRA, 15 Mb for LRA). Use `NA` to disabled length filtering.
`gaps.width`	Single integer value, alterated segments separated by a gap shorter than this value will be merged (see also 'gaps.ratio'; 500 kb for SRA, 10 Mb for LRA). Use `NA` to disabled gap filling.
`gaps.ratio`	Single numeric value, for a gap to be filled its two neighbors must be this value larger than it (see also 'gaps.width'; 1 for SRA, 1.5 for LRA). Use `NA` to disabled gap filling.
`...`	The `SRA` and `LRA` functions are only wrappers to `xRA` with distinct `lengthMax`, `lengthMin`, `gaps.width` and `gaps.ratio` values, all other arguments are passed through to `xRA`.

Value

Returns a list with a `data.frame` for each state :

`chrom`	Chromosomal location.
`inPeak`	Numeric, proportion of the sample series in the 'overlapping group'.
`overlap.start, overlap.end`	Integer, position on the chromosome for the highest peak of the SRA (region covered by the whole 'overlapping group').
`start, end`	Integer, position on the chromosome for the SRA itself (largest region covered by 2/3 of the 'overlapping group').
`extended.start, extended.end`	Integer, position on the chromosome for the extended SRA (largest region covered by 1/3 of the 'overlapping group').

Note

For Long Reccurent Abnormalities, Lenz et al. suggest to filter out regions involved in abnormal chromosome arms. For technical reasons, this filter was **NOT** implemented.

Author(s)

Sylvain Mareschal

References

Lenz G et al. "Molecular subtypes of diffuse large B-cell lymphoma arise by distinct genetic pathways". Proc Natl Acad Sci U S A. 2008 Sep 9;105(36):13520-5 (Supporting Information)

See Also

`STEPS`

`STEPS` *Selective Trends Evidenced by Penetrance Surge*

Description

This function identifies and prioritize Selective Trends Evidenced by Penetrance Surge in a CGH array series. STEPS is an alternative to the Minimal Common Region (MCR) algorithms, with the aim to identify regions frequently amplified or deleted.

Usage

```
STEPS(segPenetrance, dpen = 2, vpen = 0.8, gpen = 0.3, threshold = NA,
  nested = c("merge", "flag", "none"), digits = 3, chromEnd = FALSE, quiet = FALSE)
```

Arguments

`segPenetrance`	A `data.frame`, as a single element from the `list` returned by the `penetrance` function.
`dpen`	Single numeric value, penalty to apply to penetrance increases.
`vpen`	Single numeric value, penalty to apply to penetrance differences between wide boundaries.
`gpen`	Single numeric value, penalty to apply to genomic assymetry.
`threshold`	Single numeric value, minimum STEPS score to filter results. 0 is the less stringent threshold to use, as negative scores correspond to assymetric STEPS (ascending only on a side). Higher values will return less results (focusing on the most significant ones), however scoring and boundaries of the results will not be impacted.
`nested`	Single character value, defining how to deal with overlapping STEPS. "merge" will only keep for each set of overlapping STEPS the one with the highest score, "flag" will preserve all the STEPS but add a "nest" column with a distinct ID for each nest, and "none" won't do anything about this.
`digits`	Single integer value, to be passed to `round` for score computations.
`chromEnd`	Single logical value, whether to consider chromosome ending as a penetrance drop or not.
`quiet`	Single logical value, whether to throw diagnosis messages or not.

Details

When a specific gene alteration induces a cell selection (like in tumors), it leads to different altered fragments from a patient to an other. All these fragments have a region in common : the region containing the selecting gene (the Minimal Common Region). Such patterns can be extracted from the penetrance, as they lead to 'stairway' patterns in specific locations.

This function crawls along the penetrance from every available starting point, computing in both directions a score : a descending step grants the penetrance difference (in percents) while an ascending step penalizes by the penetrance difference multiplied by `penalty`. In each direction, the maximal score is used as boundary, and a total STEPS score for the starting point is computed as `2 * (leftMax + rightMax) - abs(leftMax - rightMax)`.

The greatest scores highlight symetric STEPS with high descending paths on both sides.

Value

Returns a subset of `segPenetrance` with the following additionnal columns :

`score`	Numeric, the two-side score for the described starting point (see 'Details').
`leftBoundary`	Integer, position considered as the left boundary of the stairway pattern.
`leftScore`	Numeric, score for the left side of the STEPS (see 'Details').
`rightBoundary`	Integer, position considered as the right boundary of the stairway pattern.
`rightScore`	Numeric, score for the right side of the STEPS (see 'Details').

Author(s)

Sylvain Mareschal

See Also

`penetrance`, `SRA`

`tk.annotate` *Interactive cghRA track annotation*

Description

This function provides a Tcl-Tk interface to annotate a region list and compute polymorphism likelihood scores.

Usage

```
tk.annotate(globalTopLevel, localTopLevel)
```

Arguments

`globalTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the top level of the embedding interface, generally a call to `tktoplevel`.
`localTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the local top level to use to build this interface, generally a `tkframe` or `ttkframe`.

Author(s)

Sylvain Mareschal

See Also

`tk.cghRA`

`tk.cghRA` *cghRA Tcl-Tk launcher*

Description

This function produces a Tcl-Tk interface merging all the cghRA components installed.

Usage

```
tk.cghRA(blocking = FALSE, tkrplot.scale = 1)
```

Arguments

`blocking`	Single logical value, whether to wait for the interface window to be closed before unfreezing the R console. The `FALSE` default let you use R and the interface in parallel, the codeTRUE is used essentially in the stand alone version.
`tkrplot.scale`	Single numeric value to be passed to `tk.modelize`.

Author(s)

Sylvain Mareschal

See Also

`tk.design`, `tk.process`, `tk.modelize`, `tk.annotate`, `tk.series`, `tk.convert`, `tk.browse`

tk.design *Interactive cghRA design processing*

Description

This function provides a Tcl-Tk interface to import a CGH array design file into a `cghRA.design` object and apply various cghRA tools on it.

Usage

```
tk.design(organism = "Human", assembly = "GRCh37",
 chromosomes = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y",
 chromFiles = "", restrictionSites = "AluI=AG|CT, RsaI=GT|AC", globalTopLevel,
  localTopLevel)
```

Arguments

`organism`	Single character value, default value for the Organism field.
`assembly`	Single character value, default value for the Assembly field.
`chromosomes`	Single character value, default value for the Chromosomes field.
`chromFiles`	Character vector, default chromosome files.
`restrictionSites`	Single character value, default value for the Restriction sites field.
`globalTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the top level of the embedding interface, generally a call to `tktoplevel`.
`localTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the local top level to use to build this interface, generally a `tkframe` or `ttkframe`.

Author(s)

Sylvain Mareschal

See Also

`tk.cghRA`, `cghRA.design`, `Agilent.design`, `custom.design` tk.modelize *Interactive copy number modelization*

Description

This function provides a Tcl-Tk interface to produce or adjust a CGH copy number model on single or multiple arrays.

Usage

```
tk.modelize(compress = "gzip", compression_level = 9, exclude = c("X", "Y", "Xp", "Xq",
  "Yp", "Yq"), globalTopLevel, localTopLevel, render = c("auto", "png", "tkrplot"),
   tkrplot.scale = 1, png.res = 100, png.file = tempfile(fileext=".png"))
```

Arguments

`compress`	To be passed to `cghRA-class` `toRdat` method.
`compression_level`	To be passed to `cghRA-class` `toRdat` method.
`exclude`	Vector, the chromosomes to exclude from the density computation and to plot with distinct symbols (use `NULL` to disable this feature). Sexual chromosomes should be excluded in heterogeneous DNA source, as their desequilibrium (2 'X' and no 'Y' in women) impact normal cells AND tumoral ones.
`globalTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the top level of the embedding interface, generally a call to `tktoplevel`.
`localTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the local top level to use to build this interface, generally a `tkframe` or `ttkframe`.
`render`	Single character value from the ones listed, defining the rendering engine for the plot. "png" is recommended and the default on any platform supporting it (needs Tcl-tk version 8.6 or higher, already available on Linux and MacOS and on Windows with R version 3.4.0 or above), and consists in displaying an export to a PNG file. "tkrplot" is more limited and kept only for backward compatibility, it relies on the external package `tkrplot` and the Windows "metafile" format. "auto" (the dzfault) will select the best engine considering to the capabilities of your installation.
`tkrplot.scale`	Single numeric value, defining a multiplying factor for plot size with the "tkrplot" engine. This argument is mainly provided to temper a bug with the "Font size multiplication factor" feature of last Windows operating system, and get plots filling the whole Tcl-tk window. As an example if you use a 150
`png.res`	Single integer value, the resolution of the plot in Pixels Per Inches. Passed to `png`, see the corresponding manual for further details. This has no effect with the "tkrplot" engine used on Windows prior to R version 3.4.0.

`png.file`	Single character value, the path to the PNG file that is displayed in the main window. The default behavior is to hide it in a temporary location, however you can define this argument to have an easier access to the images displayed in Rgb (the image will be replaced each time Rgb refresh its display). This has no effect with the "tkrplot" engine used on Windows prior to R version 3.4.0.

Details

Currently two types of files are handled: `cghRA.regions` objects exported with `saveRDT` and custom tables of segments with an optional header line describing the model.

Custom files are supposed to meet the following criteria:

- Filename extension must be ".txt".
- Table separated by tabulations, with dots as decimal separators.
- Each segment of the genome on a distinct row.
- A "chrom" column (preferably character) for segment chromosome location.
- "start" and "end" columns (1 based integers) for position on the chromosome.
- "probes" (integer) for probe amount in the segment.
- "logRatio" (numeric) for mean log-ratio of the segment.
- The first line can hold a model description, as returned by `model.test`. The line must begin with a "#" sign and describe values as "name=value" pairs separated by ", ".

Author(s)

Sylvain Mareschal

See Also

`model.auto`, `model.test`, `tk.cghRA`

`tk.series` *Interactive cghRA series processing*

Description

This function provides a Tcl-Tk interface to perform series analysis on processed arrays and designs.

Usage

```
tk.series(globalTopLevel, localTopLevel)
```

Arguments

`globalTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the top level of the embedding interface, generally a call to `tktoplevel`.
`localTopLevel`	This argument should be filled only when embedding this Tcl-Tk interface in an other. It is the local top level to use to build this interface, generally a `tkframe` or `ttkframe`.

Author(s)

Sylvain Mareschal

See Also

`tk.cghRA`

`tk.value` *Tk interface utilities*

Description

This function prompt for a single value in a Tcl-tk interface.

Usage

```
tk.value(parent = NULL, type = c("character", "integer", "double"),
  title = "Enter a value", default = "", allowEmpty = FALSE)
```

Arguments

`parent`	Tcl-tk top-level to bind the popup window to.
`type`	Single character value defining the type of the expected value.
`title`	Single character value that will be displayed as the title of the popup window.
`default`	Single value that will be used as default.
`allowEmpty`	Single logical value, whether to raise an error if the user does not provide any value or not.

Value

Returns the entered value, casted to `type`.

Author(s)

Sylvain Mareschal

trace2track — *Converts cnvScore traces to a drawable track*

Description

This function converts the data.frame trace that can be produced by `cnvScore` into a `track.table` object that can be browsed using Rgb's functions `tk.browse`. and `browsePlot`.

Usage

```
trace2track(paths, dgv.map, dgv.track)
```

Arguments

`paths`	A `data.frame`, as produced by `cnvScore` with `trace=TRUE`.
`dgv.map`	An integer matrix as returned by `map2design`, corresponding to the mapping of the polymorphism (CNV) dataset to a common design.
`dgv.track`	A `track.table`-inheriting object, the original dataset used to produce `dgv.map`.

Value

Returns a copy of `dgv.track`, in which CNVs are grouped by paths labeled with the resulting score.

Author(s)

Sylvain Mareschal

See Also

`cnvScore`, `map2design`

track.CNV.DGVsupp — *DGV supporting variant parser*

Description

This function constructs `track.CNV` objects from free annotation files provided by the Database of Genomic Variants.

It is designed to parse **supporting variants**, as opposed to `track.CNV.DGV` provided by Rgb which is designed to parse **DGV Variants**.

Usage

```
track.CNV.DGVsupp(file, name = "DGV CNV (supporting variants)", quiet = FALSE, ...)
```

Arguments

`file`	Single character value, the path to the raw file to parse. See the 'References' section below.
`name`	Single character value, the `name` field for the `track.table` object.
`quiet`	Single logical value, whether to print diagnostic `message`s or not.
`...`	Further arguments are passed to the class constructor, as a result most of the handled arguments are `track.table` arguments. Consider notably `.organism` and `.assembly` for track annotation.

Value

Return a `track.CNV` object.

Author(s)

Sylvain Mareschal

References

Example of raw file (human assembly 'hg19') : `http://dgv.tcag.ca/dgv/docs/GRCh37_hg19_supportingvariants_2014-10-16.txt`

See Also

`track.table-class`, `track.CNV-class`, `track.CNV.DGV`

`WACA` *Waves aCGH Correction Algorithm*

Description

This function applies the Waves aCGH Correction Algorithm to a series a logRatio (usually a complete series of probe logRatio from a single CGH array), using the probe-dependant biases computed by the `bias` function.

Usage

```
WACA(probeNames, probeLogRatios, bias, forceBiasOrdering = TRUE)
```

Arguments

`probeNames`	Character vector, the names of the probes to correct. All these names should be present in `bias` row.names.
`probeLogRatios`	Numeric vector, the logRatios of the probes to correct.
`bias`	A `data.frame`, as returned by the `bias` function.

`forceBiasOrdering`
: Single logical value, whether to force the `bias` data.frame ordering / subsetting / replication or not. `bias` must be ordered according to `probeNames` (that can contain duplicates), if they are not the former needs to be reordered. If they have different lengths, ordering is forced. If not, it is up to the user to assure they are or to set `forceBiasOrdering` to `TRUE` (the default value). It might be time-saving to order `bias` manually and set this parameter to `FALSE` when applying WACA on several arrays from the same design.

Value

Returns a numeric vector with the corrected logRatios, preserving the `probeNames` and `probeLogRatios` order.

Author(s)

Sylvain Mareschal

References

Lepretre F. et al. (2010) Waved aCGH: to smooth or not to smooth. Nucleic Acids Res. 2010 Apr;38(7):e94

See Also

`bias`

Index

∗Topic **classes**
    cghRA.array-class, 4
    cghRA.copies-class, 7
    cghRA.design-class, 11
    cghRA.probes-class, 15
    cghRA.regions-class, 17
    cghRA.series-class, 22
    segmentMap-class, 52
::, *48*

Agilent.design, *11*, *14*, *58*
Agilent.design *(*Design file parser*)*, 28
Agilent.probes, *17*, *50*
Agilent.probes *(*Probe file parser*)*, 47
applyMap, *26*, *34*
applyMap *(*map2design*)*, 34

bias, 2, *34*, *63, 64*
blatInstall *(*localize*)*, 32
browsePlot, *62*

cghRA.array, *4*, 4, *7*, *11*, *15*, *17*, *30*, *49*, *50*, *52*
cghRA.array-class, 4
cghRA.copies, *7*, *10, 11*, *45, 46*
cghRA.copies
    *(*cghRA.copies-constructor*)*, 10
cghRA.copies-class, 7
cghRA.copies-constructor, 10
cghRA.design, *4, 5*, *11*, *14*, *29*, *35*, *45*, *52*, *58*
cghRA.design
    *(*cghRA.design-constructor*)*, 14
cghRA.design-class, 11
cghRA.design-constructor, 14
cghRA.probes, *4, 5*, *17*, *28, 29*, *47, 48*, *50*
cghRA.probes
    *(*cghRA.probes-constructor*)*, 17
cghRA.probes-class, 15
cghRA.probes-constructor, 17
cghRA.regions, *7–10*, *17*, *21–24*, *45*, *49*, *60*
cghRA.regions
    *(*cghRA.regions-constructor*)*, 21
cghRA.regions-class, 17
cghRA.regions-constructor, 21
cghRA.series, *22*, 22, *24*
cghRA.series-class, 22
cnvScore, 24, *35*, *52, 53*, *62*
copies, 26, *36*, *39*
crossable, *5–7*, *9*, *11*, *13*, *18*, *20*
custom.design, *11*, *14*, *58*
custom.design *(*Design file parser*)*, 28
custom.probes *(*Probe file parser*)*, 47

density, *38*, *39*
Design file parser, 28
drawable, *4–13*, *18–20*
drawableFromClass.cghRA.probes, 29

envRefClass, *5–16*, *18–22*, *24*, *52, 53*

fillGaps, 30

GEDI, 31

LCN, *26*, *42*
LCN *(*copies*)*, 26
library, *48*
localize, *3*, 32
LRA *(*SRA & LRA*)*, 53

makeCluster, *50*
map2design, *25, 26*, *34*, 34, *35*, *52, 53*, *62*
message, *25*, *35*, *54*, *63*
model.apply, *27*, 35, *38*, *40–42*
model.auto, *18*, *26*, *27*, *35*, *36*, 38, *41, 42*, *60*
model.test, *26*, *35*, *36*, *40*, *41*, 41, *60*

parallelize, 44, *46, 47*
parseKaryo, 45
penetrance, *30*, *45*, 46, *55, 56*
plot, *42*

png, *59*
Probe file parser, 47
process, *15*, *17*, *21*, *47*, 48
process.mask, *48*

refTable, *7–21*
round, *3*, *26*, *44*, *55*

saveRDT, *60*
segmentMap, *25*, *35*
segmentMap-class, 52
sliceable, *5–7*, *9*, *11*, *13*, *18*, *20*
SRA, *56*
SRA (SRA & LRA), 53
SRA & LRA, 53
STEPS, *47*, *55*, 55

tk.annotate, 56, *57*
tk.browse, *57*, *62*
tk.cghRA, *11*, *15*, *17*, *57*, 57, *58*, *60*, *61*
tk.convert, *57*
tk.design, *11*, *14*, *28*, *29*, *50*, *52*, *57*, 58
tk.modelize, *57*, 59
tk.process, *15–17*, *21*, *57*
tk.process (process), 48
tk.series, *57*, 60
tk.value, 61
tkframe, *51*, *57–59*, *61*
tktoplevel, *51*, *57–59*, *61*
trace2track, *25*, *26*, 62
track.CNV, *62*, *63*
track.CNV.DGV, *62*, *63*
track.CNV.DGVsupp, 62
track.table, *7–14*, *18–21*, *34*, *35*, *52*, *62*, *63*
ttkframe, *51*, *57–59*, *61*

WACA, *3*, 63

xRA (SRA & LRA), 53